UCI-Neutrino No. 90-11
SNO-STR 90-89
13 August 1990

Data Normalization in Databases

Gerhard Bühler[1]
University of California, Irvine, CA 92717

Abstract

Data normalization is one of the primary concerns in setting up efficient databases. This report summarizes rules of data normalization. Please note the introduction.

If anything at all, perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away.
ANTOINE DE SAINT-EXUPÉRY

The rules leading to and including the third normal form can be summed up in a single statement: Each attribute must be a fact about the key, the whole key, and nothing but the key.
WIORKOWSKI AND KULL, DB2 DESIGN & DEVELOPMENT GUIDE

[1]...who does not claim copyrights for this

Introduction

We have discussed using latabases to organize the bulk of data to be generated in SNO. One of the main concerns in setting up useful databases is data normalization, i.e. the process of storing data in the most efficient way (one could probably find other definitions in textbooks on database management, but this one states it clearly enough, I think). Data normalization has specific implications on defining data keys, the "headings" which enable to cross correlate data in a database.

I have come across a poster[1] recently which is distributed by DATABASE PROGRAMMING & DESIGN, an industry journal for database managers. Since it is one of the better descriptions I have found on the subject, and since I am a basically lazy person, I will take the liberty of quoting massively from this poster throughout this report, modifying the text where I feel a need to do so. This includes the quotations on the front page.

The example given here uses the database of a kennel, its puppies, the tricks they have learned and so on, to set up a database which is refined in steps up to a very high degree of normalization. Enjoy.

The simple list

As a starting point, consider the simple list given in Figure 1. This list contains

Figure 1: Unnormalized data items for puppies

Puppy number
Puppy name
Kennel code
Kennel name
Kennel location
Trick ID 1...n
Trick name 1...n
Trick where learned 1...n
Skill level 1...n

all necessary information, but is not very useful for retrieving specific information and for maintenance of the list. In the original list of data, each puppy's description is followed by a list of tricks the puppy has learned. Some might know ten tricks, some might not know any. To answer the question, "Can Fifi roll over" we need to first find Fifi's puppy record, then scan the list of tricks associated with the record. This is awkward, inefficient, and extremely untidy.

1. Rule: Eliminate repeating groups

Moving the tricks into a separate table helps considerably. Separating the repeating groups of tricks from the puppy information results in first normal form (Figure 2). The puppy number in the trick table matches the primary key in the puppy table, providing a foreign key for relating the two tables with a join operation. Now we can answer our question with a direct retrieval: Look to see if Fifi's puppy number and the trick ID for "Roll over" appear together in the trick table.

Figure 2: First normal form

Puppy table
Puppy number
Puppy name
Kennel code
Kennel name
Kennel location

Trick table
Puppy number
Trick ID
Trick name
Trick where learned
Skill level

2. Rule: Eliminate redundant data

In the trick table, the primary key is made up of the *Puppy number* and the *Trick ID*. This makes sense for the "Where learned" and "Skill level" attributes, since they will be different for every puppy/trick combination. But the trick name depends only on the trick ID. The same name will appear redundantly every time its associated ID appears in the **Trick table**.

Suppose you want to reclassify a trick—give it a different trick ID. The change has to be made for every puppy that knows the trick! If you miss some you will have several puppies with the same trick under different IDs. This is an *update anomaly*.

Or suppose the last puppy knowing a particular trick gets run over by a steam roller. His records (of the dog, not of the steam roller) will be removed from the database, and the trick will not be stored anywhere! This is a *delete anomaly*. To avoid these problems, we need second normal form (Figure 3). To achieve this, separate the attributes depending on both parts of the key

Figure 3: Second normal form

Trick table

Puppy #	Trick ID	Trick name	Where learned	Skill level
52	27	Roll over	16	9
53	16	Nose stand	9	9
54	27	Roll over	9	5

⇓ ⇓

Tricks

Trick ID
Trick name

Puppy tricks

Puppy number
Trick ID
Trick where learned
Skill level

Puppy table

Puppy number
Puppy name
Kennel code
Kennel name
Kennel location

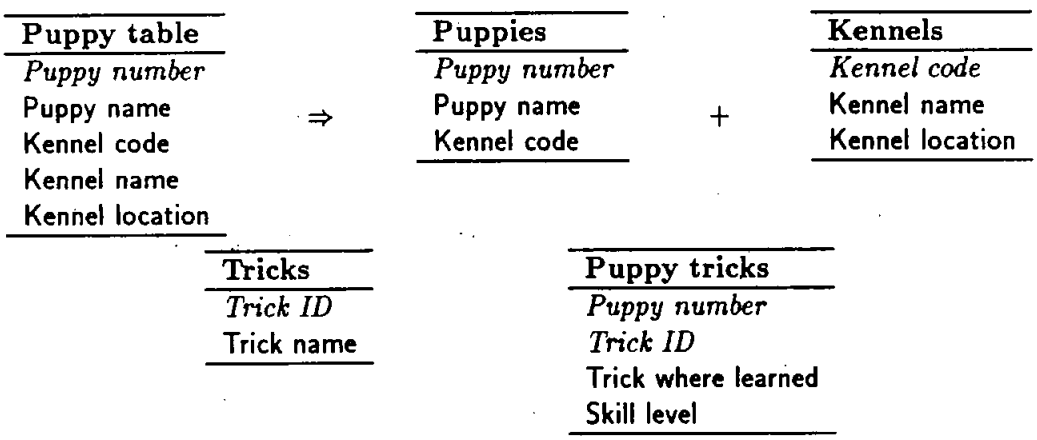
from those depending only on the trick ID. This results in two tables: **Tricks**, which gives the name for each trick ID, and **Puppy tricks**, which lists the tricks learned by each puppy.

Now we can reclassify a trick in a single operation: look up the trick ID in the "Tricks" table and change its name. The result will be instantly available throughout the application.

3. Rule: Eliminate columns not dependent on key

The puppy table satisfies first normal form—it contains no repeating groups. It satisfies second normal form since it does not have a multivalued key. But the key is puppy number, and the kennel name and kennel location describe only a kennel, not a puppy. To achieve third normal form, they must be moved into a separate table. Since they describe a kennel, *Kennel code* becomes the key of the new **Kennels** table (Figure 4). The motivation for this is the same as for

Figure 4: Third normal form



the second normal form: we want to avoid update and delete anomalies. For example, suppose no puppies from the Daisy Hill Puppy Farm were currently stored in the database. With the previous design, there would be no record of its existence! Third normal form is sufficient for most situations, but if that is not normal enough for you...

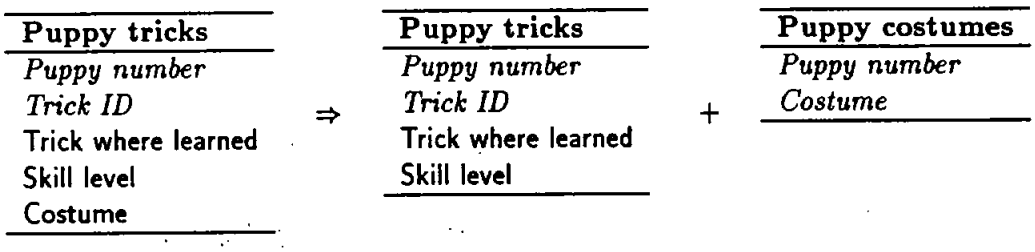
4. Rule: Isolate independent multiple relationships

This applies only to designs that include one-to-many and many-to-many relationships. An example of one-to-many is that one kennel can hold many

puppies. An example of a many-to-many relationship is that a puppy can know many tricks and several puppies might know the same trick.

Suppose we want to add a new attribute to the puppy trick table: "Costume". This way we can look for puppies that can both "sit-up-and-beg" and wear a Groucho Marx mask, for example. Fourth normal form dictates against this (against using the puppy trick table, not against begging while wearing a Groucho Marx mask). The two attributes do not share a meaningful relationship. A puppy may be able to walk upright, and it may be able to wear a wet-suit. This does not imply that it can do both at the same time (Figure 5). How will you represent this if you store both attributes in the same table?

Figure 5: Fourth normal form



5. Rule: Isolate semantically related multiple relationships

Usually, related attributes belong together. For example, if we really wanted to record which tricks each puppy could do in which costume, we would want to keep the *Costume* attribute in the puppy trick table. But there are times when special characteristics of the data make it more efficient to separate even logically related attributes.

Imagine our database will record which breeds are available in each kennel, and which breeder supplies dogs to those kennels. This suggests a Kennel-Breeder-Breed table which satisfies fourth normal form. As long as any kennel can supply any breed from any breeder, this works fine.

Now suppose a law is passed to prevent exclusive arrangements: a kennel selling any breed must offer that breed from all breeders it deals with. In other

words, if Khabul Khennels (the kennel) sells Afghans (the breed) and wants to sell any Daisy Hill Farm (the breeder) puppies, it must sell Daisy Hill Farm Afghans.

The need for fifth normal form becomes clear when we consider inserts and deletes. Suppose a kennel decides to offer three new breeds: Spaniels, Dachshunds, and West Indian Banana-Biters. Suppose further that it already deals with the breeders that can supply those breeds. This will require nine new rows in the database, one for each breeder-breed combination.

Breaking up the table reduces the number of inserts into six. Here are the tables necessary for fifth normal form, shown with the six newly inserted rows in bold type (Figure 6). If an application involves significant update activity,

Figure 6: Fifth normal form

Kennel-Breeder-Breed
Kennel number
Breeder
Breed

Kennel-Breed	
Kennel #	Breed
5	Spaniel
5	Dachshund
5	Banana-Biter

Kennel-Breeder	
Kennel #	Breeder
5	Acme
5	Puppy Factory
5	Whatapuppy

Breeder-Breed	
Breeder	Breed
Acme	Spaniel
Acme	Dachshund
Acme	Banana-Biter
Puppy Factory	Spaniel
Puppy Factory	Dachshund
Puppy Factory	Banana-Biter
Whatapuppy	Spaniel
Whatapuppy	Dachshund
Whatapuppy	Banana-Biter

fifth normal form can mean important savings. Note that these combination

tables develop naturally out of entity-relationship analysis.

References

[1] *5 Rules of Data Normalization*, poster distributed by DATABASE PROGRAMMING & DESIGN, Miller Freeman Publications, 500 Howard Street, San Francisco, CA 94105. Text by Marc Rettig, technical editor of AI EXPERT and DATABASE PROGRAMMING & DESIGN.